On the enumeration of minimal covers and minimal forbidden sets

Mathematical Programming manuscript No.
(will be inserted by the editor)

Frederik Stork · Marc Uetz

On the Enumeration of Minimal Covers and Minimal Forbidden Sets

October 22, 2002

Abstract An independence system is a family $\mathcal{I}$ of subsets of a ground set V with the property that any subset of any member of $\mathcal{I}$ also belongs to $\mathcal{I}$. The inclusion-minimal sets not in $\mathcal{I}$ are called minimal covers. We prove several complexity results related to computation, enumeration, and counting of the minimal covers of an independence system. Our motivation to study these problems is their importance in the solution of resource-constrained scheduling problems. There the minimal covers correspond to minimal subsets of jobs that must not be scheduled simultaneously, so-called minimal forbidden sets. In this context, minimal covers are the minimal infeasible $\{0,1\}$-solutions of a linear inequality system $Ax \leq b$. We propose and analyze a simple backtracking algorithm that generates a complete representation of all minimal covers of the independence system induced by a linear inequality system $Ax \leq b$. The practical performance of this algorithm is evaluated on the basis of instances from the project scheduling library PSPLIB.

Key words. independence system – minimal cover – enumeration – counting – project scheduling – resource constraints – forbidden set

1. Introduction

Given a finite ground set V, an independence system is defined as a family $\mathcal{I}$ of subsets of V with a simple hereditary property: Any subset of any member of $\mathcal{I}$ also belongs to $\mathcal{I}$. The sets belonging to $\mathcal{I}$ are called independent sets and the inclusion-minimal sets not belonging to $\mathcal{I}$ will be called the *minimal covers* of $\mathcal{I}$.

In general, it is assumed that an independence system is given by an *oracle* that decides if a set belongs to the system or not. Our first observation is that the enumeration of minimal covers of an oracle-based independence system is equivalent to the enumeration of its maximal independent sets. Hence, using a

Frederik Stork: ILOG Deutschland GmbH, Ober-Eschbacher Straße 109, D-61352 Bad Homburg, Germany. e-mail: `fstork@ilog.de`

Marc Uetz: Universiteit Maastricht, Faculty of Economics and Business Administration, Quantitative Economics, P.O. Box 616, NL-6200 MD Maastricht, The Netherlands. e-mail: `m.uetz@ke.unimaas.nl`

result of Lawler, Lenstra, and Rinnooy Kan [14], it follows that minimal covers are not P-*enumerable* in general, unless P = NP. That means that minimal covers cannot be enumerated in time polynomial in the in- and output size of the problem. The term P-*enumerable* was introduced by Valiant [25], and algorithms with polynomial time complexity in terms of in- and output are called *polynomial total time* algorithms [12]. Using previous results on counting problems by Provan and Ball [21], we also show that the computation of the *number* of minimal covers (or maximal independent sets) is a #P-complete problem. On the way, we clarify the computational complexity of several other problems that are related to the computation and/or enumeration of minimal covers.

Both the complexity and equivalence of the two enumeration problems, however, do not necessarily carry over to special cases where a particular realization of the membership-oracle is assumed. To be more precise, it is easily seen that any independence system can be explicitly described by the feasible $\{0,1\}$-solutions of a non-negative, integral linear inequality system $Ax \leq b$. The minimal covers correspond to the minimal infeasible $\{0,1\}$-vectors for $Ax \leq b$. Once such an explicit description of an independence system is given, it turns out that the enumeration problems for maximal independent sets and minimal covers are no longer equivalent. While a *quasi-polynomial incremental time* algorithm exists for the enumeration of the maximal feasible solutions, a *polynomial incremental time* algorithm for the enumeration of minimal covers can only exist if P = NP, even for a very restricted class of linear systems $Ax \leq b$ [5]. The class of polynomial incremental time algorithms is a subclass of the polynomial total time algorithms (see Definition 4). Although this leaves open the possibility of polynomial total time algorithms for both problems, it sheds a different light on the two problems: On the one hand, a quasi-polynomial total time algorithm exists for enumerating the maximally feasible solutions for $Ax \leq b$. On the other hand, in view of the hardness result of [5], the existence of such an algorithm appears not likely for enumerating the minimal covers for $Ax \leq b$.

Our original motivation to study these enumeration problems, however, is a practical application in resource-constrained (stochastic) scheduling [24]. There, minimal covers correspond to so-called *minimal forbidden sets*, subsets of jobs that must not be scheduled simultaneously either due to precedence relations or due to scarce resources required to process the jobs. The constraints for such a scheduling problem are essentially a linear inequality system $Ax \leq b$. For certain classes of branch-and-bound algorithms, a complete list of the minimally infeasible $\{0,1\}$-vectors is indeed required [24]. To practically solve this enumeration problem, we have implemented a simple backtracking algorithm. The algorithm generates a compact representation of all minimal covers of the system $Ax \leq b$ in the form of a rooted tree. For an important special case, we show that the algorithm is in fact polynomial incremental time, nicely complementing the NP-hardness result of [5]. Even though examples show that the algorithm can have a computation time exponential in terms of the in- and output size in general, a computational study with instances from the scheduling problem library PSPLIB exhibits improved performance compared to a previously suggested (also non-polynomial) divide-and-conquer approach of Lawler, Lenstra, and Rinnooy Kan [14] and Bartusch [3].

The paper is organized as follows. Section 2 addresses several complexity issues that are related to computation, enumeration, and counting of minimal covers of an arbitrary independence system. In Section 3, we introduce the independence system that arises in resource-constrained scheduling problems. We discuss several consequences of the previous section in the scheduling context, and propose and analyze a simple backtracking algorithm to enumerate all minimal covers of an arbitrary linear inequality system $Ax \leq b$. Section 4 presents our computational results with the proposed algorithm, based on scheduling instances from the PSPLIB. We conclude with some remarks in Section 5.

2. General Independence Systems

Throughout the paper let V be a finite set with $|V| = n$. The system $(V, \mathcal{I})$, $\mathcal{I} \subseteq 2^V$, is called an independence system if for any $I \in \mathcal{I}$ and any $I' \subseteq I$ we have $I' \in \mathcal{I}$. An independent set $I \in \mathcal{I}$ is maximal if there is no $I' \in \mathcal{I}$ with $I \subset I'$. The sets not belonging to $\mathcal{I}$ are called dependent sets, and a dependent set is minimal if any proper subset is independent. The inclusion-minimal dependent sets are called *minimal covers* of the independence system $(V, \mathcal{I})$, and by $\mathcal{M}$ we denote the family of all minimal covers of $(V, \mathcal{I})$.

2.1. Enumeration of Minimal Covers and Linear Defining Systems

In the most general case it is assumed that an independence system $(V, \mathcal{I})$ is given by the finite set V and an oracle $o_{\mathcal{I}}$ that decides in unit time if a given subset of V is independent or not. So $o_{\mathcal{I}} : 2^V \to \{0,1\}$, and for any $W \subseteq V$, $o_{\mathcal{I}}(W) = 1$ if and only if W is independent. We start with the following result.

Theorem 1 (Lawler, Lenstra, Rinnooy Kan [14]). *Given an arbitrary independence system $(V, \mathcal{I})$, denote by m the number of its maximal independent sets. Unless* $\mathrm{P} = \mathrm{NP}$*, there is no algorithm that enumerates the maximal independent sets of $(V, \mathcal{I})$ in time polynomial in n and m.*

The proof uses a reduction from the NP-complete problem SATISFIABILITY; see [14]. Based on this result we can also show the following.

Theorem 2. *Given an arbitrary independence system $(V, \mathcal{I})$, denote by m the number of its minimal covers. Unless* $\mathrm{P} = \mathrm{NP}$*, there is no algorithm that enumerates the minimal covers of $(V, \mathcal{I})$ in time polynomial in n and m.*

Proof. Suppose the claimed algorithm existed. Let $(V, \mathcal{I})$ be an arbitrary independence system, given by its oracle $o_{\mathcal{I}}$. Define another, dual independence system $(V, \mathcal{I}^{\mathcal{D}})$ using the following oracle $o_{\mathcal{I}^D}$: For any $W \subseteq V$, $o_{\mathcal{I}^D}(W) = 1$ if and only if $o_{\mathcal{I}}(V \setminus W) = 0$. Then $(V, \mathcal{I}^{\mathcal{D}})$ is an independence system, since $W \in \mathcal{I}^{\mathcal{D}}$ implies $W' \in \mathcal{I}^{\mathcal{D}}$ for any $W' \subseteq W$. Moreover, it follows easily that $W \subseteq V$ is a minimal cover of $(V, \mathcal{I}^{\mathcal{D}})$ if and only if $V \setminus W$ is maximal independent in $(V, \mathcal{I})$. Hence, the existence of a polynomial time enumeration algorithm for the minimal covers of $(V, \mathcal{I}^{\mathcal{D}})$ would yield a polynomial time enumeration algorithm for the maximal independent sets of $(V, \mathcal{I})$. Using Theorem 1 this is only possible if $\mathrm{P} = \mathrm{NP}$. $\square$

Using the notion of Valiant [25], Theorem 2 says that minimal covers of an arbitrary independence system are not P-enumerable, unless $\mathrm{P} = \mathrm{NP}$. Next, let us introduce an explicit description of independence systems.

Definition 1 (Linear defining system). *Given an arbitrary independence system $(V, \mathcal{I})$, and given a $m \times n$ matrix A and an m-vector b, $Ax \leq b$ is a* linear defining system *for $(V, \mathcal{I})$ if the independent sets of $(V, \mathcal{I})$ are precisely the index sets of the feasible $\{0,1\}$-solutions of $Ax \leq b$.*

It is an immediate observation that such a linear defining system always exists.

Fact 1 *Any independence system* $(V, \mathcal{I})$ *has a linear defining system* $Ax \le b$.

Proof. Given an independence system $(V, \mathcal{I})$, let $\mathcal{M} = \{M_1, \ldots, M_m\}$ be a list of all minimal covers. Denote by a^M the characteristic (row-)vector of a set $M \in \mathcal{M}$, so $a_j^M = 1$ if $j \in M$ and $a_j^M = 0$ otherwise. Then introduce for each $M \in \mathcal{M}$ one linear constraint

$$\sum_{j \in V} a_j^M \, x_j \le |M| - 1 \, .$$

If A denotes the matrix of all m (row-)vectors a^M and b is the m-vector of the corresponding right hand sides, the feasible $\{0,1\}$-solutions of $Ax \le b$ are precisely the characteristic vectors of the independent sets of $(V, \mathcal{I})$. Observe that the matrix A in this construction is even $\{0,1\}$, and $0 \le b_i \le n-1$ for the right-hand side b. ◻

In view of Fact 1, we will sometimes also use the notion of minimal covers of a linear inequality system $Ax \le b$. We then mean the minimal covers of the associated independence system, which are precisely the index sets of the minimally infeasible $\{0,1\}$-vectors for $Ax \le b$. Motivated by the construction in the proof of Fact 1, let us call a linear defining system $Ax \le b$ *boolean* if the matrix A is $\{0,1\}$.

Of course, there may be many different linear defining systems for a given independence system. One might be tempted to ask for a most 'compact' linear defining system, defining compactness as the number of rows in the linear system $Ax \le b$. For reasons that become clear shortly, let us make the following definition.

Definition 2 (Threshold dimension). *Given an arbitrary independence system* $(V, \mathcal{I})$*, define the threshold dimension* t *of* $(V, \mathcal{I})$ *as the minimum number of rows in any linear defining system* $Ax \le b$ *for* $(V, \mathcal{I})$.

By the construction in the proof of Fact 1, the threshold dimension is always upper bounded by the number of minimal covers. The case of a threshold dimension of $t = 0$ corresponds to the trivial independence system $(V, 2^V)$. In all other cases, the threshold dimension is at least 1. If the threshold dimension is exactly 1, the independence system corresponds to the feasible $\{0,1\}$-solutions of a singleton knapsack inequality. It is not hard to see that even in this case the number of minimal covers can be exponential in n, take for example the knapsack inequality $\sum_{j=1}^{n} x_j \le \lfloor n/2 \rfloor$. Moreover, Example 1 in the Appendix shows an independence system where the threshold dimension equals the number of minimal covers, and both are exponential in n. In general, it turns out that also the computation of the threshold dimension is an NP-hard problem. To this end, we need a few definitions. Given an undirected graph $G = (V, E)$, a *stable set* is a set of pairwise non-adjacent nodes of G. Obviously, the stable sets of a graph define an independence system. The threshold dimension of a graph G is defined as the minimum number t of inequalities $\sum_{j \in V} a_{kj}\, x_j \leq b_k$, $k = 1, \ldots, t$, such that for any $I \subseteq V$ it holds that I is a stable set of G if and only if the characteristic vector x^I of I fulfills all t inequalities. Graphs with a threshold dimension of 1 are called *threshold graphs*. Chvátal and Hammer [7] have shown that the computation of the threshold dimension of a graph is NP-hard in general. According to Yannakakis [26], even the decision problem if the threshold dimension of a graph is bounded by 3 is NP-complete. See the surveys [15] and [6] for more details and references[1]. With this in mind, we can show the following.

Theorem 3. *Given an arbitrary independence system $(V, \mathcal{I})$, denote by m the number of its minimal covers. Unless* P = NP*, there is no algorithm that computes the threshold dimension t of $(V, \mathcal{I})$ in time polynomial in n and m.*

Proof. If $(V, \mathcal{I})$ is the independence system induced by the stable sets of an undirected graph, the number m of minimal covers equals the number of edges of G which is bounded by n^2. Hence, if the claimed algorithm for the computation of t existed, we would be able to compute the threshold dimension of a graph in time polynomial in n. According to [7], this can only be the case if P = NP. □

2.2. Recognition and Counting of Minimal Covers

Fact 2 *Given an arbitrary independence system $(V, \mathcal{I})$, either by its oracle $o_{\mathcal{I}}$ or by some linear defining system $Ax \leq b$, and given some dependent set $U \subseteq V$. Then a minimal cover M of $(V, \mathcal{I})$ with $M \subseteq U$ can be computed in polynomial time. In particular, the recognition problem for minimal covers is solvable in polynomial time.*

[1] Threshold graphs and related questions have been considered also in the context of the so-called *PV-chunk synchronizing primitive*, which generalizes the classical *semaphore* concept for synchronization of parallel processing. In fact, apparently prior to Chvátal and Hammer [7], threshold graphs have been defined and characterized in this context by Henderson and Zalcstein [9].

Proof. A set $U \subseteq V$ is a minimal cover if and only if it is dependent and for all $j \in U$ the subsets $U \setminus \{j\}$ are independent. If $(V, \mathcal{I})$ is given by an oracle, we can verify in time $\mathrm{O}(1)$ if the given set U is indeed dependent, and in time $\mathrm{O}(n)$ if the sets $U \setminus \{j\}$ are independent. In the case that $(V, \mathcal{I})$ be given by a linear defining system $Ax \leq b$, let d be the number of rows of A. We can check in time $\mathrm{O}(dn)$ if the given set U is indeed dependent, and it requires an additional $\mathrm{O}(dn)$ time to check independence of all sets $U \setminus \{j\}$. Thus the recognition problem for minimal covers is solvable in polynomial time. If U is a dependent set but not a minimal cover, after $\mathrm{O}(n)$ removals of elements of U we obtain some minimal cover M with $M \subseteq U$. Hence, the computation of M requires $\mathrm{O}(n^2)$ time if $(V, \mathcal{I})$ is given by an oracle and $\mathrm{O}(dn^2)$ time if $(V, \mathcal{I})$ is given by a linear defining system with d rows. □

The question whether an independent set can be extended to a minimal cover turns out to be harder.

Theorem 4. *Given an arbitrary independence system $(V, \mathcal{I})$, either by its oracle $o_{\mathcal{I}}$ or by some linear defining system $Ax \leq b$, and given some independent set $U \subseteq V$. Then the decision problem wether there exists a minimal cover M of $(V, \mathcal{I})$ with $U \subseteq M$ is* NP*-complete.*

Proof. First, the problem is easily seen to be in NP. The certificate is the minimal cover M itself, and it can be verified in polynomial time by Fact 2. We will use a simple reduction of the NP-complete problem PARTITION. The problem PARTITION is the following: We are given n items of integral weight $a_j > 0$ with $2b = \sum_{j=1}^{n} a_j$, and the question is if there exists a partition of the items into two subsets of total weight b; see, e.g., [8]. Now define an independence system $(V, \mathcal{I})$ by all index sets of feasible $\{0,1\}$-solutions for the knapsack inequality $\sum_{j=0}^{n} a_j x_j \leq b$, where $a_0 := 1$. Then there exists a minimal cover M which contains the subset $U := \{0\}$ if and only if there is a solution of the given instance of the PARTITION problem. □

Let us finally address the question of computing the number of minimal covers of an independence system without explicitly enumerating them. To this end, recall the complexity class #P which was defined for counting problems in [25]. Informally, a counting problem is in #P if it can be solved by counting the number of accepting computations of a corresponding polynomial time nondeterministic Turing machine. Moreover, a counting problem is #P-hard if an oracle for its solution yields an oracle polynomial time algorithm to solve any problem in #P. A problem in #P that is #P-hard is #P-complete. For example, counting the number of Hamiltonian cycles in a graph, or counting the number of perfect matchings of a bipartite graph, are #P-complete problems. For further details see [25], [8], or [20].

Theorem 5. *Given an arbitrary independence system $(V, \mathcal{I})$, either by its oracle $o_{\mathcal{I}}$ or by some linear defining system $Ax \le b$, let m be the number of its minimal covers. Then the computation of m is* #P*-complete. This even holds if the linear defining system $Ax \le b$ is boolean.*

Proof. First it is not hard to see that the problem belongs to the class #P. Given any set $U \subseteq V$, we can verify in polynomial time if U is a minimal cover or not, using Fact 2. So consider a nondeterministic Turing machine that computes any subset $U \subseteq V$ and accepts if and only if U is a minimal cover. The number of accepting computations of this Turing machine is then exactly m. We show #P-hardness by a reduction from the problem MaxAc. MaxAc is the problem to compute a maximum cardinality anti-chain of a partially ordered set. The corresponding counting problem was shown to be #P-complete by Provan and Ball [21]. Let a partially ordered set $(V, \prec)$ be given. Any maximum cardinality anti-chain of $(V, \prec)$ equals a maximum cardinality stable set in the underlying comparability graph $G = (V, E)$, where $E = \{\{i, j\} \mid i \prec j \text{ or } j \prec i\}$. If we denote by d the maximum cardinality of any anti-chain in $(V, \prec)$, then d is computable in time polynomial in n [16]. Furthermore, let ℓ be the number of edges of G, then also ℓ is computable in time polynomial in n. Define an $(\ell + 1) \times n$ matrix A as follows. The first row of A only consists of n ones. The remaining ℓ rows are exactly the edge-node incidences of the comparability graph G. Define an $(\ell + 1)$-vector $b := (d - 1, 1, \dots, 1)$. Then there are two classes of minimal covers of $Ax \le b$, the edges of G and the maximum cardinality anti-chains of $(V, \prec)$. So if m is the number of minimal covers of $Ax \le b$, there are exactly $m - \ell$ maximum cardinality anti-chains in $(V, \prec)$. Hence, a polynomial time algorithm for the computation of the number of minimal covers would yield a polynomial time algorithm for the computation of the number of anti-chains of a partially ordered set. The claim follows. □

In fact, it is not hard to see that essentially the same proof can be used to prove #P-completeness of counting the number of maximally feasible $\{0, 1\}$-solutions of a non-negative, linear system $Ax \le b$.

Theorem 6. *Given an arbitrary independence system $(V, \mathcal{I})$, either by its oracle $o_{\mathcal{I}}$ or by some linear defining system $Ax \leq b$, let m be the number of its maximally feasible $\{0,1\}$-solutions. Then the computation of m is* #P*-complete. This even holds if the linear defining system $Ax \leq b$ is boolean.*

2.3. More on the Enumeration of Minimal Covers

Given an independence system by some linear system $Ax \leq b$, we could not prove that the polynomial total time enumeration of minimal covers is hard. However, a recent paper of Boros, Elbassioni, Khachiyan, and Makino [5] sheds some light on this question.

Proposition 1 (Boros et al. [5]). *Consider a linear system $Ax \leq b$, with $\{0,1\}$-matrix A and non-negative b where all but one of the components of b equal 1. Let $\mathcal{X}$ be a subset of the minimal covers $\mathcal{M}$ for $Ax \leq b$. Then it is* NP*-complete to decide if $\mathcal{M} \setminus \mathcal{X} \neq \emptyset$.*

The proof relies on a reduction from the NP-complete problem Stable Set for undirected graphs. We include it here for the sake of completeness.

Proof. Membership in NP follows easily, since any element in $\mathcal{M} \setminus \mathcal{X}$ serves as certificate that can be verified in time polynomial in the input size. Given an arbitrary undirected graph $G = (V, E)$ and an integer t, the problem Stable Set asks if the graph contains a stable set of cardinality at least t or not. So define a $\{0,1\}$-matrix A as the edge-node incidence matrix for G, appended with one more row consisting of n ones. Define the right hand side b as a vector of $|E|$ ones, with the $(|E|+1)$st component $t-1$. Then the minimal covers $\mathcal{M}$ for $Ax \leq b$ are exactly the edges E of G and the stable sets of G with cardinality t (if such stable sets exist). Given $\mathcal{X} = E$, deciding wether $\mathcal{M} \setminus \mathcal{X} \neq \emptyset$ amounts to decide – in time polynomial in the encoding length of G – if the graph G has a stable set of size at least t or not. □

The consequence of this result is that a *polynomial incremental time* algorithm cannot exist for the enumeration of minimal covers for $Ax \leq b$. To this end, we need two more definitions.

Definition 3 (Increments problem). *Given an implicit description of a family $\mathcal{M} \subseteq 2^V$, and let $\mathcal{X} \subseteq \mathcal{M}$ be given, then either find a new element $M \in \mathcal{M} \setminus \mathcal{X}$ or decide that $\mathcal{M} \setminus \mathcal{X} = \emptyset$.*

Now we can define the class of polynomial incremental time algorithms.

Definition 4 (Polynomial incremental time). *The enumeration problem for a family $\mathcal{M} \subseteq 2^V$, with an implicit description of encoding length d, is solvable in polynomial incremental time if the increments problem can be solved in time polynomial in d and $|\mathcal{X}|$, for any $\mathcal{X} \subseteq \mathcal{M}$.*

It is obvious that any polynomial incremental time enumeration algorithm is also a polynomial total time enumeration algorithm: $\mathcal{M}$ can be generated by iteratively solving the increments problem, starting with $\mathcal{X} = \emptyset$. After $|\mathcal{M}| + 1$ iterations, each of which runs in time polynomial in terms of the in- and output, the complete family $\mathcal{M}$ is generated. However, the converse need not be true, and despite of the hardness result of Proposition 1, a polynomial total time algorithm could exist for enumerating the minimal covers of a linear system $Ax \leq b$. We conjecture that such an algorithm does not exist unless P = NP.

Without going into further details, we mention that the increments problem for maximally feasible solutions for a non-negative system $Ax \leq b$ is solvable in *quasi-polynomial time* [5]. That means that the time complexity is in the order $t^{\mathrm{o}(\log t)}$, where t denotes the input size. This yields a quasi-polynomial incremental time algorithm for enumerating maximally feasible solutions, which contrasts the situation for minimal covers.

3. Minimal Covers and Resource-Constrained Project Scheduling

In this section, we 'specialize' to our original motivation to study the enumeration problem for minimal covers, namely an independence system that arises in resource-constrained project scheduling. It turns out, however, that this does not constitute any loss of generality.

3.1. Resource-Constrained Project Scheduling

Assume that a set $V = \{1, 2, \ldots, n\}$ of jobs has to be executed subject to both precedence and resource constraints. Precedence constraints are given in the form of a partial order $\prec$ on V, so $(V, \prec)$ is a partially ordered set. Whenever $i \prec j$, processing of j cannot be started before i has been completed. We denote by $G = (V, E)$ the comparability graph associated to $(V, \prec)$, with edges $E = \{\{i, j\} \mid i \prec j \text{ or } j \prec i\}$, and let $\ell \in \mathrm{O}(n^2)$ be the number of edges of G. In addition to the precedence constraints, the jobs need different resource types $k = 1, \ldots, d$ while being processed. A constant amount of $b_k \geq 0$ units of each resource type is available and each job j consumes $0 \leq a_{kj} \leq b_k$ units of resource k while in process. The jobs are to be scheduled such that all precedence constraints are respected and at any time the total resource consumption of the jobs in process does not exceed the resource availability b_k, for each resource type k. Let us make the following definition.

Definition 5 (Minimal forbidden set). *Given a resource-constrained project scheduling problem as described above, a subset U of jobs is called a forbidden set if*

(i) U is an anti-chain with respect to the partial order $(V, \prec)$, and

(ii) there exists some resource type k with $\sum_{j \in U} a_{kj} > b_k$.

If U is a forbidden set, but no proper subset of U is forbidden, then U is a minimal forbidden set.

Let us define $\mathcal{F} \subseteq 2^V$ as the family of all minimal forbidden sets, and denote by $f = |\mathcal{F}|$ the number of minimal forbidden sets. Minimal forbidden sets already appeared in [23], and they form the basis of numerous algorithmic approaches to solve resource-constrained project scheduling problems. For example, they are used to derive cutting planes for integer programming formulations, e. g. in [1,18]. In stochastic scheduling where job processing times are uncertain, minimal forbidden sets are the staring point to define certain scheduling policies, e. g. in [11,10,17,19,24]. Finally, also branch-and-bound algorithms have been defined on the basis of minimal forbidden sets, e.g. in [10,4,24].

If $\mathcal{I} \subseteq 2^V$ denotes the subsets of jobs that can be feasibly scheduled at the same time, $(V, \mathcal{I})$ is obviously an independence system. The subsets $I \in \mathcal{I}$ are sometimes also called *feasible* subsets of jobs. The minimal covers $\mathcal{M}$ of this independence system are the minimal subsets of jobs that cannot be processed simultaneously. These are either minimal forbidden sets or precedence related pairs of jobs $\{i, j\} \in E$. A linear defining system $Ax \leq b$ for this independence system is obtained as follows. The matrix A is defined by the resource requirements a_{kj} for all $k = 1, \ldots, d$ and all $j = 1, \ldots, n$, and the right-hand sides are the resource availabilities b_k for all $k = 1, \ldots, d$. Moreover, for each edge $e = \{i, j\}$ of the comparability graph G, define one additional row e of A, where $a_{ei} = a_{ej} = 1$, $a_{ek} = 0$ for all $i \neq k \neq j$, and let the corresponding right hand side $b_e = 1$. This way, the additional ℓ rows are the edge-node incidence matrix of G. With these definitions, the feasible $\{0, 1\}$-solutions of the linear system $Ax \leq b$ are precisely the independent sets of the above defined independence system $(V, \mathcal{I})$. Minimal covers are either precedence related pairs of jobs or minimal forbidden sets, and there are exactly $m = f + \ell$

minimal covers. While $\ell \in O(n^2)$, the same examples as in Section 2 show that f, the number of minimal forbidden sets, can be exponential in n, the number of jobs. Whenever there are no precedence constraints, minimal forbidden sets and minimal covers coincide. Hence, the consideration of minimal forbidden sets instead of minimal covers does not constitute any loss of generality. The complexity results of Section 2 now have the following consequences.

Corollary 1. *Given a resource-constrained project scheduling problem with n jobs and d resource types. Then*

(i) the computation of the number f of minimal forbidden sets is #P-complete,

(ii) given a set U of jobs, one can decide in time polynomial in n and d if U is a minimal forbidden set,

(iii) given a set U of jobs, it is NP-complete to decide wether there exists a minimal forbidden set F with $U \subseteq F$.

Moreover, the following lemma will be used later.

Lemma 1. *Given a resource-constrained project scheduling problem with n jobs and d resource types, and given a set U of jobs. Then one can decide in time polynomial in n and d if there exists a forbidden set F with $U \subseteq F$.*

Proof. First we have to verify that U is an anti-chain with respect to the partially ordered set $(V, \prec)$. This can be done in $O(n^2)$ time. Then define $a_k(U) := \sum_{j \in U} a_{kj}$. If $a_k(U) > b_k$ for some k, U itself is forbidden. Otherwise we have to check if there exists a resource type k and an anti-chain $F \supseteq U$ such that $\sum_{j \in F} a_{kj} > b_k$. To this end, let $W \subseteq V \setminus U$ be the set of all jobs $j \in V$ that are not precedence related to any job in U. In other words, for any $i \in W$ and any $j \in U$, neither $i \prec j$ nor $j \prec i$. Now, for each resource type k consider the induced subgraph $G_k(W)$ with nodes W, edges $\{\{i,j\} \in E \mid i,j \in W\}$, and node weights a_{kj}, $j \in W$. If and only if the maximum weight stable set in $G_k(W)$ exceeds $b_k - a_k(U)$ for some resource type k, there exists a forbidden set F with $U \subseteq F$. But a maximum weight stable set in $G_k(W)$ can be computed in time polynomial in n by solving a minimum flow problem, due to the fact that $G_k(W)$ is a comparability graph; see [16]. Hence, the overall time complexity is polynomial in n and d and the claim is proved. $\square$

3.2. Enumerating Minimal Forbidden Sets

In this section, we describe a simple backtracking algorithm which enumerates the minimal forbidden sets $\mathcal{F}$ for an arbitrary instance of the resource-constrained project scheduling problem. Notice that the algorithm applies as well to the enumeration of the minimal covers $\mathcal{M}$ of an arbitrary linear system $Ax \leq b$, since the minimal covers exactly correspond to the minimal forbidden sets whenever no precedence constraints are present.

In the special case that the scheduling problem is described by one resource type only, we show that the algorithm can be turned into a polynomial incremental time algorithm. In particular, it enumerates minimal forbidden sets in polynomial time in terms of n and f, the in- and output size of the problem.

3.2.1. Description of the Algorithm The basic approach is to enumerate subsets of V in a tree T where each node u of T, except the root node, is associated to exactly one job $j \in V$ (however, the mapping of nodes of the tree to jobs is not an injection). If node u is associated to some job $j \in V$, u has a child node for each job $i = j+1, \ldots, n$. The root node has a child node for each job $i \in V$. Each node u of the tree, associated to some job j, defines a subset $U \subseteq V$ with $j \in U$ by traversing the tree from u to the root node and collecting the associated jobs on that path. In fact, a node of the tree only consists of its associated job j and a pointer to its father. With these definitions, there is a one-to-one correspondence between the set of nodes of T and the power set 2^V of all subsets of V.

To build a tree $T(\mathcal{F})$ which exactly represents all minimal forbidden sets $\mathcal{F}$, the tree T is fathomed during this generic process like in a branch-and-bound algorithm. A node u is discarded as soon as it can be proved that neither U nor any superset of U that is located in the subtree rooted at u is a minimal forbidden set. This happens, for example, as soon as there are two jobs $i, j \in U$ with $i \prec j$ or $j \prec i$. If U itself is a minimal forbidden set, the node u is stored as a leaf of the tree $T(\mathcal{F})$. If U is an anti-chain of $(V, \prec)$ and not forbidden, there may exist minimal forbidden sets F with $U \subset F$ that are located in the subtree rooted at u, hence branching is required on u. If some node does not represent a minimal forbidden set, and does not have any further descendants, it is deleted from the tree. (Notice that the deletion has to be done recursively which requires to additionally store the number of children in each node of the tree.) The details can be found in Algorithms 1 and 2.

Let us give a simple example. Let $V = \{1,2,3,4\}$, and let there be only one precedence constraint $1 \prec 2$. So the comparability graph $G = (V, E)$ has only one edge $\{1,2\}$. There is one resource type with availability $b_1 = 3$, and the resource requirement of jobs is $a_{11} = 3$, $a_{12} = 2$, $a_{13} = 1$, and $a_{14} = 1$. The linear defining system for the independence system of jobs that can be scheduled simultaneously is

$$\begin{pmatrix} 3 & 2 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 3 \\ 1 \end{pmatrix}.$$

The minimal forbidden sets are $\{1,3\}$, $\{1,4\}$, and $\{2,3,4\}$. Together with $\{1,2\}$ these are the minimal covers of the above linear defining system. Figure 1 depicts the trees T and $T(\mathcal{F})$. Every leaf of $T(\mathcal{F})$ corresponds to a minimal forbidden set.

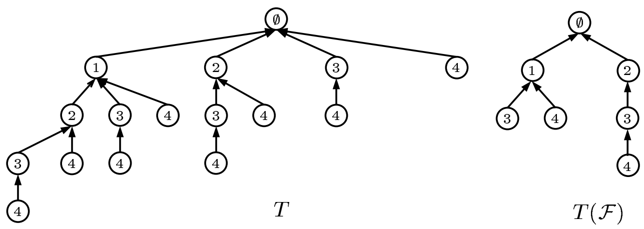

Figure 1. Example of the trees T and $T(\mathcal{F})$.

3.2.2. Analysis of the algorithm for one resource type Let us briefly discuss the computational complexity of the proposed algorithm for the special case that there is only one resource type. We prove that the algorithm can be implemented to run polynomial in n and f, the size of the in- and output. Hence, for the case of one resource type, the minimal forbidden sets are P-enumerable.

Let us assume that $d = 1$, so there is only one resource type. For ease of notation, denote by a_j be the resource consumption of jobs $j \in V$. The idea is to consider the jobs in a non-increasing order of their resource consumption a_j, so assume that $a_1 \geq a_2 \geq \cdots \geq a_n$. Then the following observation is immediate.

Lemma 2. *If the jobs are considered in non-increasing order of resource requirements a_j, then each forbidden set U found by the generic procedure described in Section 3.2.1 is already minimal forbidden.*

Proof. Say a node u of the tree is generated that corresponds to a forbidden set $U = \{j_1, j_2 \ldots, j_t\}$, where $j_1 < j_2 < \cdots < j_t$. Then, by construction, the set $U \setminus \{j_t\}$ is not forbidden, and since $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_t}$, also none of the sets $U \setminus \{j_i\}$ is forbidden, for all $i = 1, \ldots, t-1$. □

Now recall Lemma 1 which yields that for any given subset of jobs $U \subseteq V$, we can decide in time polynomial in n if U is contained in some (not necessarily minimal) forbidden set or not. In other words, one can decide in time polynomial in n if some node u of the tree T will eventually lead to some forbidden set or not. Combined with Lemma 2, we now obtain the following.

Lemma 3. *If the jobs are considered in non-increasing order of their resource requirements a_j, at any node u of the tree T considered in the generic procedure described in Section 3.2.1, one can decide in time polynomial in n if node u will eventually lead to some minimal forbidden set or not.*

Since the number of nodes in $T(\mathcal{F})$ is linear in n and f, this shows that the time required to compute the tree $T(\mathcal{F})$ is polynomial in n and linear in f.

Theorem 7. *Let a resource-constrained project scheduling problem with n jobs and one resource type be given, and let f be the number of minimal forbidden sets. Then there exists an algorithm that enumerates the minimal forbidden sets in time polynomial in n and f. In other words, minimal forbidden sets are* P-*enumerable.*

In fact, this result does not come as a surprise, since it was already show in [14] that the maximal independent sets of a knapsack inequality are P-enumerable. Moreover, the minimal covers of a knapsack inequality $\sum_{j \in V} a_j\, x_j \leq b$ are precisely the maximal independent sets of the knapsack inequality $\sum_{j \in V} a_j\, x_j \leq b'$, with $b' = \sum_{j \in V} a_j - b - 1$. The situation of Theorem 7 is only slightly more general in that there are also precedence constraints, so in the corresponding linear defining system $Ax \leq b$, the matrix A consists of one knapsack inequality and the edge-node incidences of a comparability graph. If precedence constraints are absent, we are back at one knapsack inequality. In this case our algorithm has a time complexity of $\mathrm{O}(n^2 f)$. In fact, this exactly matches the complexity given in [14].

To see that the above algorithm even yields a polynomial incremental time algorithm, suppose a subset $\mathcal{X}$ of the minimal forbidden sets $\mathcal{F}$ is given. Modify the algorithm in such a way that any node u that corresponds to a minimal forbidden set in $\mathcal{X}$ is fathomed as well. To examine a single node thus takes time polynomial in n and $|\mathcal{X}|$. If the tree is searched in a depth-first fashion, after examination of at most n^2 nodes either a new element $F \in \mathcal{F} \setminus \mathcal{X}$ is found or an element $F \in \mathcal{X}$ is found. In the latter case we have to backtrack. The number of backtracks is obviously bounded by $|\mathcal{X}|$, so after examination of at most $|\mathcal{X}|\, n^2$ nodes a new element $F \in \mathcal{F} \setminus \mathcal{X}$ is found or it can be concluded that $\mathcal{F} \setminus \mathcal{X} = \emptyset$. Hence we obtain:

Theorem 8. *Let a resource-constrained project scheduling problem with n jobs and one resource type be given, and let a subset $\mathcal{X}$ of the minimal forbidden sets $\mathcal{F}$ be given. Then there exists an algorithm that either computes a new element $F \in \mathcal{F} \setminus \mathcal{X}$ or decides that $\mathcal{F} \setminus \mathcal{X} = \emptyset$, in time polynomial in n and $|\mathcal{X}|$.*

At first sight this seems to contradict the hardness result of Proposition 1, since the system $Ax \leq b$ induced by a scheduling problem with one resource type has exactly the same form as in the NP-hardness proof of the proposition (even if each job requires exactly one unit of the resource). However, the difference is that the precedence constraints define a *comparability graph*, and for comparability graphs the stable set problem, that was used in the proof of Proposition 1, is no longer NP-hard.

For more than one resource type, the described algorithm is not polynomial in terms of f, the number of minimal forbidden sets. The reason is that, given a node u, we can no longer decide in polynomial time if the associated set of jobs U is contained in a *minimal* forbidden set or not. (Recall Corollary 1 (iii)). Hence, more nodes than eventually necessary are possibly evaluated by our algorithm. The number of such nodes may be exponential in terms of f, as demonstrated by Example 2 in the Appendix.

3.2.3. Implementation of the algorithm: Fathoming heuristics Contrary to what was described in Section 3.2.2, in our implementation we only considered heuristic but very efficient algorithms in order to decide if a node of the tree potentially leads to a minimal forbidden set or not. These simple tests greatly improved the performance of the simple generic procedure described in Section 3.2.1. To simplify notation, we henceforth omit the resource index k. By a_j we denote the vector of resource requirements of job j, so $a_j = (a_{1j}, \ldots, a_{dj})$. The resource supply is denoted by $b = (b_1, \ldots, b_d)$, and any inequality involving a_j or b is meant component-wise. Moreover, a_U denotes the vector of total resource consumption of any subset U of jobs, so $(a_U)_k = \sum_{j \in U} a_{kj}$, $k = 1, \ldots, d$.

First, motivated by the results of Section 3.2.2, also for instances with more than one resource type it showed to be computationally more effective to consider the jobs in a suitable ordering: Therefore we identify a resource type k that is 'scarce' in the following sense: k as a resource type with smallest ratio $b_k / \sum_{j \in V} a_{kj}$. So we assume that the jobs are numbered in non-increasing order of their consumption of this resource type k.

Next, two jobs i and j cannot be in a common minimal forbidden set if there is a precedence constraint between i and j, so either $i \prec j$ or $j \prec i$. In addition, we implemented two other heuristic tests to determine if no minimal forbidden set contains both i and j. First, if the resources required by i and j are *disjoint* in the sense that $a_{ki} \cdot a_{kj} = 0$ for all resource types k, then i and j together do not belong to any minimal forbidden set. Second, let U be the set of jobs that are unrelated to both i and j with respect to the precedence constraints. Then, if $a_i + a_j + a_U \leq b$, then i and j are not contained in a common minimal forbidden set either. All above tests are performed as preprocessing, and the resulting information is stored in a Boolean matrix $Y = (y_{ij})_{i,j \in V}$ of size $n \times n$ in order to provide access in O(1) time. So whenever $y_{ij} = 1$ for two jobs i and j, then there exists no minimal forbidden set F with $i \in F$ and $j \in F$.

Finally, we implemented another heuristic which is particularly useful if resource constraints are weak. For a given node u, associated to some job j and a set U of jobs with $j \in U$, sum up the resource requirements of all jobs $W \subseteq \{j+1, \ldots, n\}$ that are not precedence-related to any of the jobs in U. Then if $a_U + a_W \leq b$, the subtree emanating from node u can be discarded because there cannot be any forbidden set in this subtree. Notice that $U \cup W$ is not necessarily an anti-chain, so it is not guaranteed that the subtree emanating from node u contains a minimal forbidden set even if $a_U + a_W \not\leq b$. We also experimented with the exact method which checks if there exists a forbidden set F with $U \subseteq F$ (recall that this is possible in polynomial time by Lemma 1). However, the computational overhead to solve the associated minimum-flow problems was too large. Algorithms 1 and 2 show further details of the algorithm.

Algorithm 1: Compute all minimal forbidden sets

Input : Partially ordered set $(V, \prec)$, resource constraints $(a_j)_{j \in V}, b$.
Output: The set of minimal forbidden sets represented by the tree $T(\mathcal{F})$.

choose $k \in \{1, \ldots, d\}$, create ordering L of jobs with non-increasing a_{kj};
compute matrix $Y = (y_{ij})_{i,j \in V}$;   // $y_{ij} = 1$ if $\nexists\, F \in \mathcal{F}$ with $i, j \in F$
$\mathcal{F} := \emptyset$;   // stores the minimal forbidden sets
root := root node of tree $T(\mathcal{F})$; *Stack* := $\emptyset$;
for *all jobs* $j \in V$ **do**
    $u :=$ CreateNode(j, *root*);
    push u on *Stack*;
while *Stack* $\neq \emptyset$ **do**
    remove node u from *Stack*;
    $j :=$ job associated to u; $U :=$ set of jobs associated to U;
    for *all jobs* $i >_L j$ **do**
        EvaluateNode(i, u);
        if $U \cup \{i\}$ *is a minimal forbidden set* **then**
            $w :=$ CreateNode(i, u) and add w to $\mathcal{F}$;
        if $U \cup \{i\}$ *is feasible* **then**
            $w :=$ CreateNode(i, u) and add w to *Stack*;
    (recursively) delete u if it is not min. forbidden and has no children;
return $\mathcal{F}$;

Algorithm 2: EvaluateNode

```
Input  : A set U of jobs (represented by node u) and a new job i.
Output: Status of the set U ∪ {i}.
if y_ij = 1 for some job j ∈ U then
    return can be discarded;
if a_U + a_i ≰ b then
    for j ∈ U do
        if a_U + a_i − a_j ≰ b then
            return can be discarded;
    return minimal forbidden;
W := {j ∈ V | j >_L i, y_jl = 0 ∀ l ∈ U ∪ {i}};
// U ∪ {i} ∪ W could contain F ∈ 𝓕 with (U ∪ {i}) ⊂ F
if a_U + a_i + a_W ≤ b then return can be discarded;
else return feasible;
```

4. Computational Results

We first describe the computational setup and the benchmark instances, and then analyze the performance of the proposed algorithm in dependence on different parameters which have been used to generate the instances. This analysis is particularly of interest from the scheduling perspective, since it provides additional insights into the nature of the instance test sets from the PSPLIB. Finally, we compare the performance of our algorithm with a previously suggested divide-and-conquer approach.

4.1. Setup and Benchmark Instances

Our experiments were conducted on a Sun Ultra 1 with 143 MHz clock pulse operating under Solaris 2.7. The code is written in C++ and has been compiled with the GNU g++ compiler version 2.91.66 using the -O3 optimization option. The memory limit was set to 50 MB.

We have tested the proposed algorithm on instances of the library PSPLIB [22] that was generated by Kolisch and Sprecher with the help of the instance generator ProGen [13]. The library contains instances with 30, 60, 90, and 120 jobs, respectively. The instances have been generated by modifying three parameters, (i) the *network complexity* (NC) which is the average number of direct successors of a job[2], (ii) the *resource factor* (RF) which is the average number of different resource types required to process a job divided by the total number of resource types, and (iii) the *resource strength* (RS), which is a measure of the scarcity of the resources. The latter parameter is in the interval $[0, 1]$, and the closer to 0, the scarcer are the resources; see [13] for details. The parameters have been chosen out of the sets $NC \in \{1.5, 1.8, 2.1\}$ and $RF \in \{0.25, 0.5, 0.75, 1.0\}$. The number of different resource types is 4 for all instances. For the benchmark sets with 30, 60 and 90 jobs the resource strength RS has been chosen from the values $\{0.2, 0.5, 0.7, 1.0\}$, while for instances with 120 jobs it was chosen from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. For each combination of the parameters, 10 instances have been generated at random. This results in 480 instances for each of instance sizes 30, 60, and 90, and 600 instances with 120 jobs. Notice that, on average, the resources are scarcer for the instances with 120 jobs. Such instances are known to be particularly hard with respect to optimization.

Before we turn to our computational experiences with these instances, let us briefly comment on the relationship between the systems of minimal forbidden sets and the above mentioned parameters. According to Radermacher [23, p. 237], instances are *essentially equal* if both precedence constraints and the systems of minimal forbidden sets coincide. In this respect, the variation of the resource factor RF does not necessarily lead to essentially different instances: Although two instances have a different resource factor, they can be identical in the sense that they have identical precedence constraints and systems of minimal forbidden sets. For example, if there is only one resource type and each job requires this resource, the resource factor is obviously 1. However, the same system of minimal forbidden sets $\mathcal{F}$ can be represented by $|\mathcal{F}|$ different resource types, which generally leads to a resource factor strictly smaller than 1. Another remark addresses the above definition of network complexity. Since the definition as the average number of direct successors disregards transitive precedence constraints, instances with identical network complexity may have an essentially different topology, hence also essentially different systems of minimal forbidden sets. For example, for $V = \{1, \dots, 4\}$, the precedence constraints given by the directed arcs $A_1 := \{(1,2), (1,4), (3,4)\}$ and $A_2 := \{(1,2), (2,3), (3,4)\}$ both have a network complexity

[2] The partial order $(V, \prec)$ is defined by a directed acyclic graph $G = (V, A)$ for these instances, and $i \prec j$ whenever there is a directed path from i to j in G.

$NC = 3/4$. While $(V, \prec_2)$ is a chain, and hence has no non-trivial anti-chain, $(V, \prec_1)$ has three non-trivial anti-chains. Nevertheless, our computational results with the PSPLIB instances show that, on average, there is a meaningful correlation between all three parameters and the system of minimal forbidden sets.

4.2. Computational Study

Table 1 shows for each test set the number of solved instances (#solved), that is, all minimal forbidden sets could be computed within the memory restriction of 50 MB, as well as the average and maximum number of minimal forbidden sets ($\varnothing\ |\mathcal{F}|$ and max. $|\mathcal{F}|$) and required computation times ($\varnothing$ CPU and max. CPU). As the table suggests, the algorithm easily computes all minimal forbidden sets for the instances with 30 jobs; the computation time is negligible. Most of the instances with 60 jobs can also be solved in short time, however, there already exist few (17) instances for which not all minimal forbidden sets could be determined within the memory restriction of 50 MB (even with a limit of 500 MB, 7 instances remain unsolved). Although for larger instances the average memory requirement increases, the algorithm solves more than a half of the instances with 90 and 120 jobs with no more than 50 MB memory requirement. Even for instances with 120 jobs, for all instances with scarce resources ($RS = 0.1$) or small resource factor ($RF = 0.25$, that is, each job requires only one resource type on average), the algorithm computes all minimal forbidden sets at an average running time of less than 5 seconds.

Table 1. For each set of instances the table displays the number of instances (#inst.), the number of solved instances (#solved), the average and the maximum number of minimal forbidden sets ($\varnothing\ |\mathcal{F}|$ and max. $|\mathcal{F}|$), and the average and the maximum computation time in seconds ($\varnothing$ CPU and max. CPU).

#jobs	#inst.	#solved	$\varnothing\ \|\mathcal{F}\|$	max. $\|\mathcal{F}\|$	$\varnothing$ CPU	max. CPU
30	480	480	326	4,411	0.01	0.2
60	480	463	101,773	2,163,692	7	167
90	480	309	255,476	1,867,239	23	490
120	600	340	243,871	1,996,505	13	200

Minimal Forbidden set statistics. Figures 2 and 3 show how the average number and cardinality of minimal forbidden sets depend on the instance parameters RS, RF, and NC. Since we did not observe that these parameters were significantly correlated, all figures are based on average values with respect to the whole set of instances with 30 jobs. As expected, both the number and cardinality of minimal forbidden sets heavily depend on the instance parameters RS, RF, and NC. Let us briefly analyze the outcome of this evaluation.

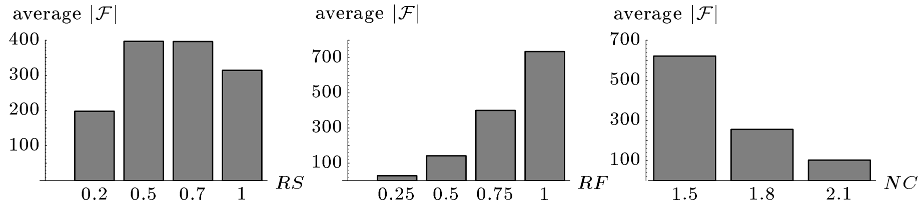

Figure 2. The plots display the average number of minimal forbidden sets depending on the instance parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

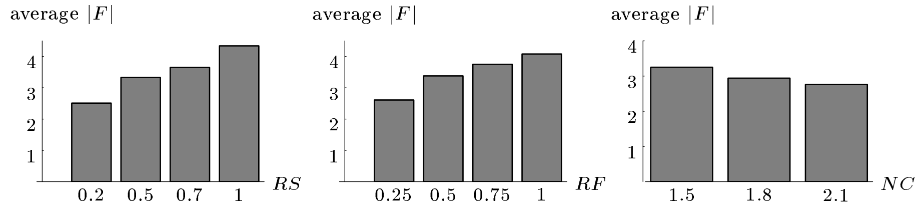

Figure 3. The plots display the average cardinality of minimal forbidden sets depending on the instance parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

The dependence of the average cardinality of minimal forbidden sets $|F|$ on the resource strength RS as shown in Figure 3 is intuitive; the scarcer the resources the smaller are the minimal forbidden sets on average. With respect to the average number of minimal forbidden sets in dependence of the resource strength RS, it is noticeable that this figure is small either if the resource strength RS is very low (0.2; scarce resources) or very high (1.0; loose resource constraints). For scarce resources, this is due to the fact that the minimal forbidden sets tend to be small, hence there are fewer on average. This is also confirmed by Figure 3. For loose resources, already many anti-chains tend to be feasible, hence there are fewer minimal forbidden sets at all, with larger cardinality on average. This is again confirmed by Figure 3.

The behavior of the average cardinality of minimal forbidden sets in dependence of the resource factor RF in Figure 3 can be explained as follows. If each job requires only one or few resource types on average, that is, the resource factor RF is small, it is very likely that in a given anti-chain there are pairs (i, j) of jobs with disjoint resource requirements ($a_{ik} \cdot a_{jk} = 0$ for all resource types k), hence minimal forbidden sets tend to be smaller on average. Consequently, there are also fewer of them, as can be seen in Figure 2.

With respect to the network complexity NC, our results show that both number and cardinality of minimal forbidden sets trends down when the network complexity NC increases. The reason is that, for the considered instances, the total number of precedence constraints (including transitive ones) increases with the network complexity. Recall, however, that the network complexity is not a measure for the total number of precedence constraints in general.

We finally observed that the average cardinality of the minimal forbidden sets increases with the number of jobs. The respective average values, based on the number of solved instances as given in Table 1, are 3.5 (maximum 10) for 30 jobs, 4.9 (maximum 16) for 60 jobs, 5.1 (maximum 13) for 90 jobs, and 4.5 (maximum 12) for 120 jobs. Notice that the average and maximum cardinality is comparatively small for the test set with 120 jobs, which is due to the fact that the resource strength parameters are smaller for these instances, and perhaps also since 260 of the 600 instances could not be solved within our 50 MB memory limitation.

Computation times. Let us next analyze the computational performance with respect to running times. Table 2 first shows the average and maximal computation times for our algorithm, both with and without the fathoming heuristics. For the version without fathoming heuristics, only the given precedence constraints have been taken into account to define the Boolean matrix Y (see Section 3.2.3).

Table 2. For both versions of the algorithm, the table displays the number of solved instances (#solved) and the respective average and the maximum computation time in seconds (∅ CPU and max. CPU); based on instances with 30 and 60 jobs.

	#jobs	#solved	∅ CPU	max. CPU
with fathoming	30	480	0.01	0.2
no fathoming	30	480	0.04	0.5
with fathoming	60	463	7	167
no fathoming	60	446	145	6,280

The results confirm that the fathoming heuristics proposed in Section 3.2.3 are worthwhile. Figure 4 shows more details with respect to the computation times, based on the test set with 30 jobs. It is intuitive that the computation times are small whenever there are only few, and small minimal forbidden sets, and large if there are many and large minimal forbidden sets. This is indeed validated by Figure 4.

There is however, one more remark on the computation times which concerns the fathoming heuristics proposed in Section 3.2.3. Without these heuristics, the dependence of computation time on the resource factor RF gives a picture which is exactly reverse, showing that these heuristics are extremely effective particularly for instances where the resource factor RF is small. In fact, for $RF = 0.25$, the computation time decreases from 41 ms (without fathoming) to 1.4 ms (Figure 4; with fathoming).

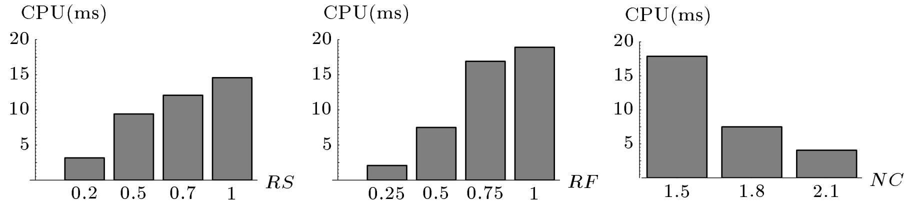

Figure 4. The plots display the average running time (in 1/1000 sec.) depending on the parameters RS (left), RF (middle), and NC (right). The data is based on the test set with 30 jobs per instance.

Divide-and-conquer algorithms. We have also experimented with a divide-and-conquer algorithm, based on an approach suggested by Bartusch [3]. While [3] enumerates minimal forbidden sets in the context of resource-constrained scheduling problems, an essentially equivalent divide-and-conquer approach was previously suggested by Lawler, Lenstra, and Rinnooy Kan in [14]. Their algorithm enumerates the minimal covers of an arbitrary inequality system. In terms of scheduling, the basic idea is as follows. Partition the given instance, say I, into d partial instances $I_1, \ldots, I_d$ where each I_k only consists of jobs which require a positive amount of resource type k, $k = 1, \ldots, d$. Then, for each I_k, the set of minimal forbidden sets $\mathcal{F}_k$ is calculated with respect to resource type k only. This has the advantage that each of the subproblems can be solved in polynomial time with respect to its in- and output, which is n and $|\mathcal{F}_k|$; see Theorem 7. On the other hand, the systems of minimal forbidden sets $\mathcal{F}_k$ for the subproblems I_k may be exponential in terms of $\mathcal{F}$ itself, and the efficient computation of the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$ constitutes a non-trivial problem in its own. Based on the instances from the PSPLIB we have compared the time required to compute $\mathcal{F}$ using the algorithm proposed in this paper with the overall time required time to compute all minimal forbidden sets $\bigcup_k \mathcal{F}_k$ for all partial instances I_k, $k = 1, \ldots, d$. It turned out that these computation times are in fact comparable on average, however, for only few instances the divide-and-conquer approach was more efficient. In particular, this comparison does not yet take into account the additional overhead required to compute the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$.

In fact, using a straightforward implementation (see, e.g. [14, Sect. 4.6] for an approach how to avoid duplication), this overhead turned out to be a major bottleneck of the divide-and-conquer approach; it required far more computation time than the computation of the sets $\bigcup_k \mathcal{F}_k$ itself. Hence, a divide-and-conquer approach seems to be beneficial only for instances with very particular structure, e. g. Example 2 in the Appendix.

Memory requirements. Finally, it is obvious that the data structure given by the tree $T(\mathcal{F})$ is much more compact in comparison to an ordinary list representation of $\mathcal{F}$, where each minimal forbidden set is stored as a list of job numbers. In our experiments, the memory requirement could be reduced by 50% on average.

5. Concluding Remarks

We addressed several complexity questions that are related to the enumeration of minimal covers of an independence system. The main remaining open question is a proof or falsification of the conjecture that the minimal covers of an arbitrary inequality system $Ax \leq b$ are P-enumerable only if P = NP. Nevertheless, for practical purposes the algorithm we proposed is simple and can be of interest also in other applications than scheduling. Moreover, at least on the scheduling instances we have used for our computational study, the algorithm improves significantly upon the divide-and-conquer approach that was previously suggested both in [14] and [3].

Acknowledgements. We thank Marc Pfetsch for several enlightening discussions and for pointing us to the paper [14]. The second author also likes to thank Alexander Grigoriev for helpful discussions. Parts of this work were done while the authors were with the Technische Universität Berlin, Germany. There, the first author was supported by the Deutsche Forschungsgemeinschaft (DFG), grant Mo 446/3-3, and the second author was supported by the Bundesministerium für Bildung und Forschung (bmb+f), grant 03-MO7TU1-3, and the German-Israeli Foundation for Scientific Research and Development (GIF), grant I 246-304.02/97.

References

1. R. Alvarez-Valdés Olaguíbel and J. M. Tamarit Goerlich. The project scheduling polyhedron: Dimension, facets, and liftig theorems. *European Journal of Operational Research*, 67:204–220, 1993.

2. E. Balas and E. Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34:119–148, 1978.
3. M. Bartusch. *Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln.* PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, Germany, 1984.
4. M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.
5. E. Boros, K. Elbassioni, L. Khachiyan, and K. Makino. Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities. *SIAM Journal on Computing*, 31:1624–1643, 2002.
6. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia (PA), 1999.
7. V. Chvátal and P. L. Hammer. Aggregation of inequalities in integer programming. *Annals of Discrete Mathematics*, 1:145–162, 1977.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.
9. P. B. Henderson and Y. Zalcstein. A graph-theoretic characterization of the $\mathrm{PV}_{\mathrm{chunk}}$ class of synchronizing primitives. *SIAM Journal on Computing*, 6:88–108, 1977.
10. G. Igelmund and F. J. Radermacher. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13:29–48, 1983.
11. G. Igelmund and F. J. Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13:1–28, 1983.
12. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
13. R. Kolisch and A. Sprecher. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.
14. E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9:558–565, 1980.
15. N. V. R. Mahadev and U. N. Peled. *Threshold Graphs and Related Topics*, volume 56 of *Annals of Discrete Mathematics.* North-Holland, Amsterdam, 1995.
16. R. H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Order*, NATO Advanced Science Institute Series, pages 41–101. D. Reidel Publishing Company, Dordrecht, 1985.
17. R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems I: General strategies. *ZOR - Zeitschrift für Operations Research*, 28:193–260, 1984.
18. R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. Technical Report 680, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, 2000. Submitted.
19. R. H. Möhring and F. Stork. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 52:501–515, 2000.
20. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading (MA), 1994.

21. J. S. Provan and M. O. Ball. The complexity of counting cuts and of the probability that a graph is connected. *SIAM Journal on Computing*, 12:777–788, 1983.

22. PSPLIB. `ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/`, 2000.

23. F. J. Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4:227–252, 1985.

24. F. Stork. *Stochastic Resource-Constrained Project Scheduling.* PhD thesis, Department of Mathematics, Technische Universität Berlin, Berlin, Germany, 2001.

25. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

26. M. Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods*, 3:351–358, 1982.

A. Appendix

Example 1. Let $V = \{1, \ldots, 4n\}$, and let $V_1 = \{1, \ldots, 2n\}$ and $V_2 = \{2n+1, \ldots, 4n\}$ be a partition of V. Now, for each $U_1 \subseteq V_1$ define a corresponding $U_2 := \{u + 2n \mid u \in U_1\}$, and let $\mathcal{M} := \{U_1 \cup U_2 \mid U_1 \subset V_1, |U_1| = n\}$ be the minimal covers of an independence system $(V, \mathcal{I})$. In other words, all subsets $I \subseteq V$ that do not contain any of the sets in $\mathcal{M}$ as subset belong to $\mathcal{I}$. Obviously, this is an independence system.

Here, the number of minimal covers is $\binom{2n}{n} \in \Omega(2^n)$ by definition. Now for any two distinct minimal covers, say $U_1 \cup U_2$ and $W_1 \cup W_2$, where $U_1, W_1 \in V_1$ and $U_2, W_2 \in V_2$ according to the above definition, two different linear inequalities are required, since otherwise at least one of the sets $U_1 \cup W_1$, $U_2 \cup W_2$, $U_1 \cup W_2$, or $U_2 \cup W_1$ would not be independent. Hence, at least $\Omega(2^n)$ inequalities are required in any linear defining system of $(V, \mathcal{I})$. In other words, the threshold dimension of $(V, \mathcal{I})$ is $\Omega(2^n)$. □

Example 2. Let $V = \{1, \ldots, n\}$, $d = 2$, $b_1 = b_2 = 2n - 4$, $a_{1,1} = a_{1,2} = a_{2,n-1} = a_{2,n} = n$, and $a_{kj} = 1$ otherwise.

Then $\mathcal{F}$ consists of exactly six sets, namely $\{1, 2\}$, $\{n-1, n\}$, and $\{i, 3, 4, \ldots, n-3, n-2, j\}$ for $i = 1, 2$ and $j = n-1, n$. Hence $|\mathcal{F}| \in \mathrm{O}(1)$ for any $n \in \mathbb{N}$, but the number of nodes which are examined within our backtracking algorithm is exponential in n. Notice that this is an example where the described divide-and-conquer approach by Bartusch (see Section 3.2) is beneficial, since it runs polynomial in n. □